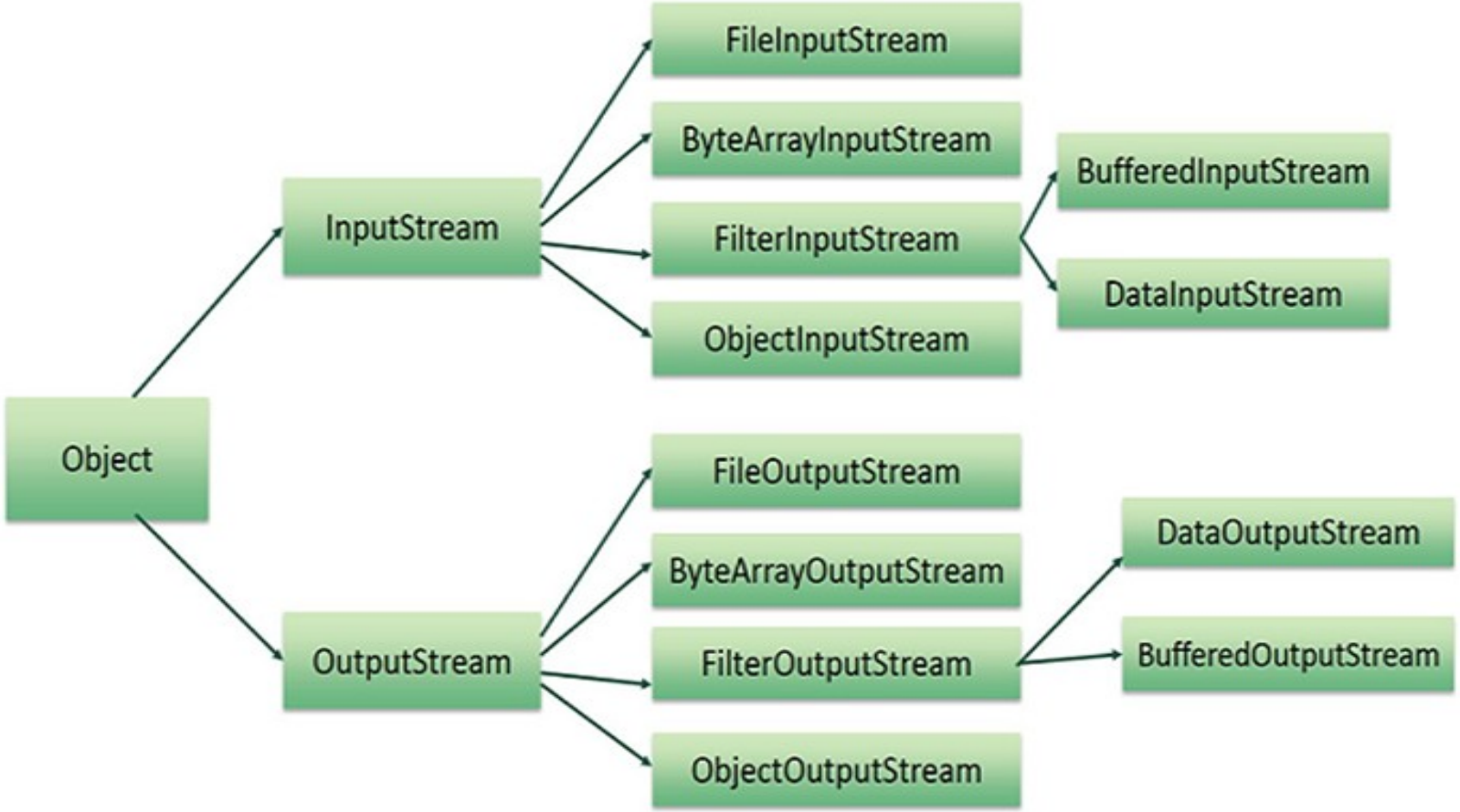
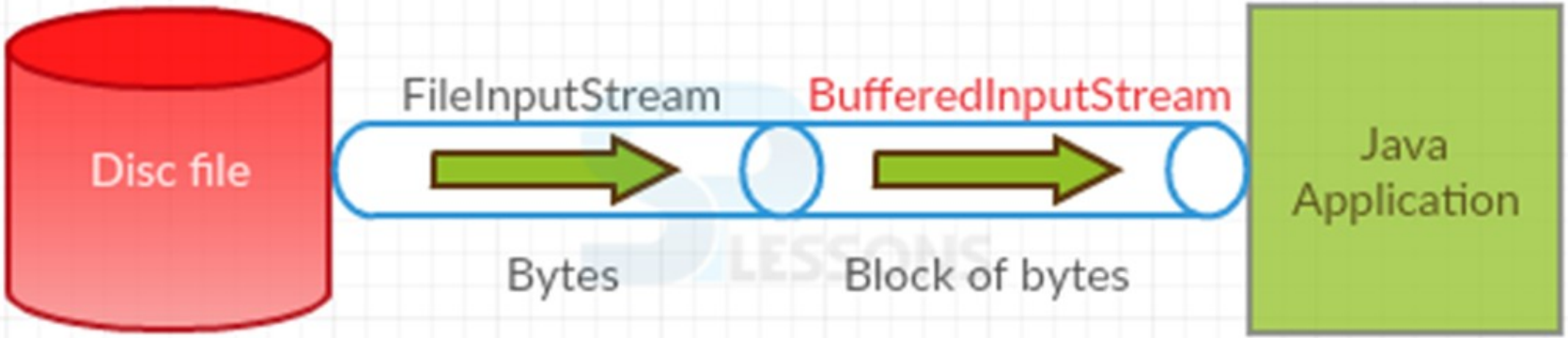
Занятие 8:
Пакет java.io и работа с ресурсами

Иерархия пакета io



Принцип работы с потоками в Java



Принцип работы с потоками в Java

InputStream и **OutputStream** – абстрактные классы, определяющие базовый ввод и вывод для потоков данных.

URI — символьная строка, позволяющая идентифицировать какой-либо ресурс: документ, изображение, файл, службу, ящик электронной почты и т. д. Прежде всего, речь идёт, конечно, о ресурсах сети Интернет и Всемирной паутины. URI предоставляет простой и расширяемый способ идентификации ресурсов. Расширяемость URI означает, что уже существуют несколько схем идентификации внутри URI, и ещё больше будет создано в будущем.

URL — это URI, который, помимо идентификации ресурса, предоставляет ещё и информацию о местонахождении этого ресурса. A URN — это URI, который только идентифицирует ресурс в определённом пространстве имён

ByteArrayInput/Output

Класс **ByteArrayInputStream** представляет входной поток, использующий в качестве источника данных массив байтов. Он имеет следующие конструкторы:

- ✓ ByteArrayInputStream(byte[] buf)
- ✓ ByteArrayInputStream(byte[] buf, int offset, int length)

В качестве параметров конструкторы используют массив байтов buf, из которого производится считывание, смещение относительно начала массива offset и количество считываемых символов length.

Класс **ByteArrayOutputStream** представляет поток вывода, использующий массив байтов в качестве места вывода.

- ✓ ByteArrayOutputStream
- ✓ ByteArrayOutputStream()
- ✓ ByteArrayOutputStream(int size)

Первая версия создает массив для хранения байтов длиной в 32 байта, а вторая версия создает массив длиной size.

FileInput/Output Stream

Классы **FileInputStream** /**Output** Предназначены для чтения/записи байтов из/в файл

Для создания объекта FileOutputStream используется конструктор, принимающий в качестве параметра путь к файлу для записи. Для записи строки ее надо сначала перевести в массив байтов. Строка записывается в файл с помощью метода write.

Классы **FileInputStream** и **FileOutputStream** предназначены прежде всего для записи двоичных файлов.

BufferedInput/Output

Для оптимизации операций ввода-вывода используются буферизуемые потоки. Эти потоки добавляют к стандартным специальный буфер в памяти, с помощью которого повышается производительность при чтении и записи потоков.

Класс **BufferedInputStream** просто оптимизирует производительность при работе с потоком **ByteArrayInputStream**.

Класс **BufferedOutputStream** в конструкторе принимает в качестве параметра объект **OutputStream** - в данном случае это файловый поток вывода **FileOutputStream**. И также производится запись в файл. Опять же **BufferedOutputStream** не добавляет много новой функциональности, он просто оптимизирует действие потока вывода.

DataOutputStream

Классы **DataOutputStream** и **DataInputStream** позволяют записывать и считывать данные примитивных типов.

- ✓ **writeBoolean**(boolean v) : записывает в поток булевое однобайтовое значение
- ✓ **writeByte**(int v): записывает в поток 1 байт, которые представлен в виде целочисленного значения
- ✓ **writeChar**(int v): записывает 2-байтовое значение char
- ✓ **writeDouble**(double v): записывает в поток 8-байтовое значение double
- ✓ **writeFloat**(float v): записывает в поток 4-байтовое значение float
- ✓ **writeInt**(int v): записывает в поток целочисленное значение int
- ✓ **writeLong**(long v): записывает в поток значение long
- ✓ **writeShort**(int v): записывает в поток значение short
- ✓ **writeUTF**(String str): записывает в поток строку в кодировке UTF-8

DataInputStream

Класс DataInputStream действует противоположным образом - он считывает из потока данные примитивных типов. Соответственно для каждого примитивного типа определен свой метод для считывания:

- ✓ **boolean readBoolean()**: считывает из потока булевое однобайтовое значение
- ✓ **byte readByte()**: считывает из потока 1 байт
- ✓ **char readChar()**: считывает из потока значение char
- ✓ **double readDouble()**: считывает из потока 8-байтовое значение double
- ✓ **float readFloat()**: считывает из потока 4-байтовое значение float
- ✓ **int readInt()**: считывает из потока целочисленное значение int
- ✓ **long readLong()**: считывает из потока значение long
- ✓ **short readShort()**: считывает значение short
- ✓ **String readUTF()**: считывает из потока строку в кодировке UTF-8
- ✓ **int skipBytes(int n)**: пропускает при чтении из потока n байтов

ObjectOutputStream

Применяются для сериализации/десериализации объектов

- ✓ void **close**(): закрывает поток
- ✓ void **flush**(): очищает буфер и сбрасывает его содержимое в выходной поток
- ✓ void **write**(byte[] buf): записывает в поток массив байтов
- ✓ void **write**(int val): записывает в поток один младший байт из val
- ✓ void **writeBoolean**(boolean val): записывает в поток значение boolean
- ✓ void **writeByte**(int val): записывает в поток один младший байт из val
- ✓ void **writeChar**(int val): записывает в поток значение типа char, представленное целочисленным значением
- ✓ void **writeDouble**(double val): записывает в поток значение типа double
- ✓ void **writeFloat**(float val): записывает в поток значение типа float
- ✓ void **writeInt**(int val): записывает целочисленное значение int
- ✓ void **writeLong**(long val): записывает значение типа long
- ✓ void **writeShort**(int val): записывает значение типа short
- ✓ void **writeUTF**(String str): записывает в поток строку в кодировке UTF-8
- ✓ void **writeObject**(Object obj): записывает в поток отдельный объект

ObjectInputStream

- ✓ void **close**(): закрывает поток
- ✓ int **skipBytes**(int len): пропускает при чтении несколько байт, количество которых равно len
- ✓ int **available**(): возвращает количество байт, доступных для чтения
- ✓ int **read**(): считывает из потока один байт и возвращает его целочисленное представление
- ✓ boolean **readBoolean**(): считывает из потока одно значение boolean
- ✓ byte **readByte**(): считывает из потока один байт
- ✓ char **readChar**(): считывает из потока один символ char
- ✓ double **readDouble**(): считывает значение типа double
- ✓ float **readFloat**(): считывает из потока значение типа float
- ✓ int **readInt**(): считывает целочисленное значение int
- ✓ long **readLong**(): считывает значение типа long
- ✓ short **readShort**(): считывает значение типа short
- ✓ String **readUTF**(): считывает строку в кодировке UTF-8
- ✓ Object **readObject**(): считывает из потока объект

Сериализация/десериализация

Существует два основных интерфейса для сериализации/десериализации:

- ✓ Serializable
- ✓ Externalizable

Первый – маркерный интерфейс. Второй требует реализации методов readExternal и writeExternal

Работа с файлами до Java 7

```
BufferedReader reader = null;
try {
    reader = new BufferedReader(
                new InputStreamReader(
                    new FileInputStream(FILE_NAME), Charset.forName("UTF-8")));
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    // log error
} finally {
    if (reader != null) {
        try {
            reader.close();
        } catch (IOException e) {
            // log warning
        }
    }
}
```

Try-with-resource

```
try (BufferedReader reader = new BufferedReader(
        new InputStreamReader(
                new FileInputStream(FILE_NAME), StandardCharsets.UTF_8))){
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    // log error
}
```

Scanner

Scanner - класс, позволяющий упростить работу с потоками ввода-вывода благодаря более высокоуровневой организации. Это надстройка над рассмотренными ранее классами.

Конструктор Сканера в качестве аргумента получает поток ввода-вывода, который должен обрабатываться сканером.

Основные методы класса Scanner позволяют получать следующую запись необходимого типа в файле, а также проверять ее наличие.